CHUYÊN ĐỀ
TỔ CHỨC DỮ LIỆU
MÔ HÌNH DỮ LIỆU TUYẾN TÍNH

Giảng viên: VŨ QUỐC HOÀNG
(vqhoang@fit.hcmus.edu.vn)

Nội dung trình bày

- Danh sách
- Các thao tác cơ bản trên danh sách
- Tập hợp và Túi
- Cài đặt danh sách
- Tìm kiếm trên danh sách
- Sắp xếp danh sách

Danh sách (List)

- Danh sách là một dãy hữu hạn gồm không hoặc nhiều phần tử cùng loại (cùng kiểu)
- Danh sách không có phần tử nào được gọi là danh sách rỗng, kí hiệu []
- Danh sách có *n* phần tử lần lượt là a_1, a_2, …, a_n được kí hiệu là [a_1, a_2, …, a_n]:
  - *n* gọi là chiều dài của danh sách: danh sách rỗng có chiều dài là 0
  - a_i ($1 \leq i \leq n$) được gọi là phần tử thứ *i* của danh sách
- Danh sách hình thành một thứ tự trước sau (quan hệ tuyến tính) trên các phần tử nên là một mô hình dữ liệu tuyến tính

Danh sách

- Nếu các phần tử của danh sách có kiểu là *T* thì kiểu của danh sách là list-of-*T*, *T* có thể là kiểu bất kì (có thể là một kiểu danh sách khác):
  - Danh sách các số nguyên tố nhỏ hơn 20
    - [2, 3, 5, 7, 11, 13, 17, 19]
  - Danh sách các ngày trong tuần
    - [Mon, Tue, Wed, Thur, Fri, Sat, Sun]
  - Danh sách sinh viên
  - Một dòng chữ là một danh sách các kí tự
  - Một trang sách là một danh sách các dòng chữ

Danh sách

- Các đặc trưng của danh sách:
  - Các phần tử cùng kiểu
  - Các phần tử có thứ tự
  - Các phần tử có thể trùng (ở thứ tự khác nhau)
- Hai danh sách được xem là bằng nhau khi:
  - Cùng chiều dài
  - Các phần tử giống nhau ở cùng thứ tự

Danh sách

- Danh sách là một cấu trúc đệ qui
- Một danh sách kiểu list-of-*T* có thể được định nghĩa đệ qui như sau:
  - *Cơ sở*: [] là một danh sách list-of-*T*
  - *Đệ qui*: nếu *H* là một đối tượng kiểu *T* và *R* là một danh sách list-of-*T* thì ghép *R* vào sau *H* được một danh sách list-of-*T*, kí hiệu [*H* | *R*]; *H* gọi là phần tử đầu, *R* gọi là phần sau của danh sách
- Chẳng hạn [1, 2, 3] = [1 | [2 | [3 | []]]]

Các thao tác cơ bản trên danh sách

- Mô hình dữ liệu danh sách hỗ trợ các thao tác cơ bản:
  - Tạo danh sách rỗng
  - Kiểm tra danh sách rỗng
  - Truy cập phần tử của danh sách
  - Thêm phần tử vào danh sách
  - Xóa phần tử khỏi danh sách
  - Sửa phần tử của danh sách

Các thao tác cơ bản trên danh sách

- Thao tác truy cập không làm thay đổi danh sách
  - Danh sách tĩnh chỉ hỗ trợ duy nhất thao tác này: chẳng hạn Từ điển
- Danh sách truy cập ngẫu nhiên:
  - cho phép truy cập phần tử bất kì của danh sách với thời gian truy cập như nhau
- Danh sách truy cập hạn chế:
  - Truy cập phần tử đầu
  - Truy cập phần tử cuối
  - Truy cập phần tử sau/trước một phần tử nào đó

Các thao tác cơ bản trên danh sách

- Các thao tác thêm/xóa/sửa làm thay đổi danh sách
  - Danh sách động là danh sách hỗ trợ các thao tác này
  - Phụ thuộc vào khả năng truy cập danh sách
    - Ngăn xếp (Stack):
      - Thêm vào đầu (Push)
      - Xóa ở đầu (Pop)
      - Truy cập ở đầu (Top)

      → Cơ chế LIFO (Last In First Out)
    - Hàng đợi (Queue):
      - Thêm vào cuối (Enqueue)
      - Xóa ở đầu (Dequeue)
      - Truy cập ở đầu (Front), cuối (Rear)

      → Cơ chế FIFO (First In First Out)

Các thao tác cơ bản trên danh sách

- Vì chúng ta có thể xem xét danh sách theo hai góc nhìn
  - thủ tục
  - khai báo
- Nên các thao tác trên danh sách có thể cài đặt theo hai kĩ thuật tương ứng là
  - lặp
  - đệ qui

Các thao tác cơ bản trên danh sách

- Thao tác tính chiều dài của danh sách *L*, len(*L*), có thể được tính bằng kĩ thuật lặp như sau:
  - Bước 1: len = 0
  - Bướ 2: Lặp lại Bước 2 trong khi danh sách *L* khác rỗng

    len = len + 1

    xóa phần tử đầu khỏi danh sách *L*
  - Bước 3: len(*L*) = len

Các thao tác cơ bản trên danh sách

- Thao tác tính chiều dài của danh sách *L*, len(*L*), có thể được tính bằng kĩ thuật đệ qui như sau:
  - *Cơ sở*: len([]) = 0
  - *Đệ qui*: len([*H* | *R*]) = 1 + len(*R*)

len([1, 2, 3])? → 3

len([2, 3])? → 2

len([3])? → 1

len([])? → 0

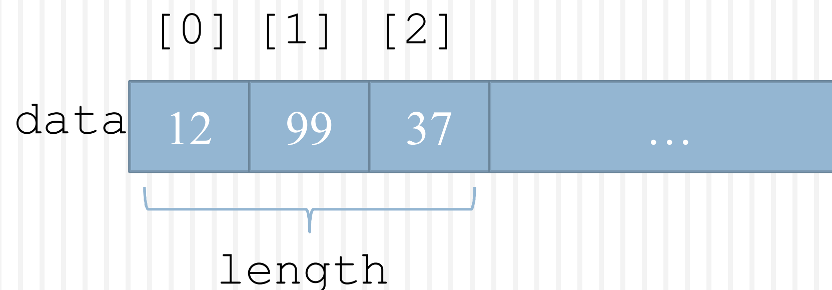
Tập hợp và Túi (Set and Bag)

- Tập hợp là mô hình dữ liệu cho phép quản lý nhiều phần tử:
  - cùng kiểu
  - không có thứ tự
  - không trùng nhau
- Túi là mô hình dữ liệu cho phép quản lý nhiều phần tử:
  - cùng kiểu
  - không có thứ tự
  - có thể trùng nhau

Cài đặt danh sách Mảng (Array)

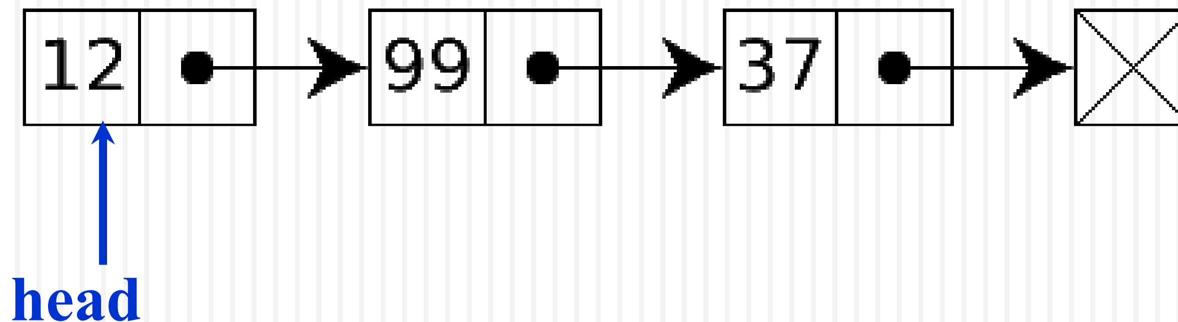

```
//cấp phát tĩnh
struct LIST
{
  DATATYPE data[MAX];
  int length;
};
```

```
//cấp phát động
struct DYNLIST
{
  DATATYPE *data;
  int length;
};
```



Cài đặt danh sách
Danh sách liên kết (Linked-list)


```
struct NODE
{
   DATATYPE data;
   NODE *next;
};
struct LIST
{
   NODE *head;
};
```



Cài đặt danh sách

- Cài đặt danh sách bằng mảng:
  - Thứ tự được cài đặt ngầm định bằng chỉ số
  - Thao tác truy cập rất hiệu quả: truy cập ngẫu nhiên
  - Các thao tác thêm, xóa kém hiệu quả
  - Sử dụng bộ nhớ kém hiệu quả
- Cài đặt danh sách bằng danh sách liên kết:
  - Thứ tự được cài đặt tường minh bằng con trỏ
  - Thao tác truy cập kém hiệu quả: truy cập tuần tự
  - Các thao tác thêm, xóa rất hiệu quả
  - Sử dụng bộ nhớ hiệu quả

Tìm kiếm trên danh sách
Tìm kiếm tuần tự (Linear Search)

- Tìm một phần tử có khóa cho trước trong một danh sách bằng cách kiểm tra lần lượt các phần tử theo thứ tự cho đến khi tìm được hoặc hết danh sách
  - Thời gian chạy xấu nhất: tỉ lệ với chiều dài của danh sách
  - Trong trường hợp danh sách bị hạn chế truy cập như trong danh sách liên kết thì đây là phương pháp tìm kiếm duy nhất

Tìm kiếm tuần tự


```
int LinearSearch(int a[], int n, int k)
{
   for(int i = 0; i < n; i++)
        if(a[i] == k)
               return i;
   return -1;
}
int SentinelSearch(int a[], int n, int k)
{
   a[n] = k;
   for(int i = 0; ; i++)
        if(a[i] == k)
               return i;
}
```

Tìm kiếm tuần tự với "lính canh"

Tìm kiếm trên danh sách
Tìm kiếm nhị phân (Binary Search)

- Tìm một phần tử có khóa *k* trong một danh sách đã được xắp xếp tăng theo khóa:
  - Xét phần tử ở giữa danh sách
    - Nếu có khóa bằng *k*: tìm thấy
    - Nếu có khóa nhỏ hơn *k*: tìm qua nửa bên phải
    - Nếu có khóa lớn hơn *k*: tìm qua nửa bên trái
  - Danh sách truy cập ngẫu nhiên
  - Rất hiệu quả: ~log*n*

Tìm kiếm nhị phân


```
int BinarySearch(int a[], int k, int l, int r)
{
    if(l > r)
        return -1;
    int m = (l + r)/2;
    if(a[m] == k)
        return m;
    if(a[m] < k)
        return BinarySearch(a, k, m + 1, r);
    else
        return BinarySearch(a, k, l, m - 1);
}
```

Sắp xếp danh sách

- Sắp xếp một danh sách $[a_1, a_2, \ldots, a_n]$ là thay danh sách đó bằng một hoán vị của nó $[b_1, b_2, \ldots, b_n]$ sao cho $b_1 \leq b_2 \leq \ldots \leq b_n$, với ≤ là một thứ tự sắp xếp nào đó
- Một trường hợp đơn giản là sắp xếp tăng trên một danh sách các số nguyên
  - Danh sách ban đầu: [2, 4, 2, 1, 5]
  - Danh sách sau khi sắp xếp tăng: [1, 2, 2, 4, 5]

Sắp xếp danh sách Quick Sort

- QuickSort là một thuật toán sắp xếp hay dùng
  - Hiệu quả (*n*log*n*)
  - Cài đặt đơn giản bằng đệ qui
- Ý tưởng
  - Danh sách rỗng: đã được sắp
  - Danh sách không rỗng:
    - Chia thành 2 danh sách con [b_1, …b_k] và [c_1, …, c_l] sao cho mọi phần tử trong danh sách *b* đều không lớn hơn mọi phần tử trong danh sách *c*
    - Sắp xếp (đệ qui) 2 danh sách *b*, *c*
    - Nối danh sách *b* sau danh sách *c*

Sắp xếp danh sách
Quick Sort


```
void quicksort(LIST *&l)
{
   if(isempty(l))
         return;
   LIST *less = createlist(); LIST *greater = createlist();
   int pivot = removefirst(l);
   while(!isempty(l))
   {
        int x = removefirst(l);
        if(x < pivot)
             addfirst(less, x);
        else
             addfirst(greater, x);
   }
   quicksort(less);  quicksort(greater);
   addfirst(greater, pivot); l = appendlist(less, greater);
}
```

24

Hỏi và Đáp